MINI58 FMC AP2AP

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	Jump between two independent applications
BSP Version	Mini58 Series CMSIS BSP v3.00.003
Hardware	NuTiny-EVB-MINI58-LQFP48 V1.0 board

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Function Description

1.1 Introduction

The NuMicro Mini58™ series is equipped with 16K/32K bytes APROM for application. This example shows two independent applications located on different address of APROM can jump to each other.

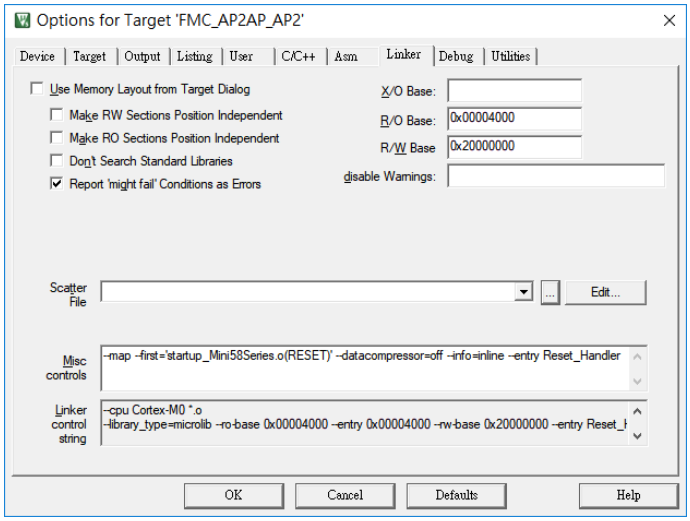
In this example, the R/O (Read-Only) base of application 1 is 0x0, while the R/O base of application 2 is 0x4000. By Arm® Cortex™-M0 architecture, the interrupt vector table must be located at memory address 0x0 for CPU to trap the exceptions. In case of application 2 whose execution address is 0x4000, an exception can make CPU fetch the unexpected interrupt vector because address 0x0 is not the code area of application 2. To solve this issue, NuMicro Mini58™ FMC supports “Vector Remap” command, which can remap one APROM page to memory address 0x0. In this way, application 2 runs at address 0x4000 can have interrupts trapped by CPU.

1.2 Principle

This example includes two Keil® projects, FMC_AP2AP_AP1.uvproj and FMC_AP2AP_AP2.uvproj for building the two applications.

The user must build FMC_AP2AP_AP2.uvproj first to generate image FMC_AP2AP_AP2.bin, otherwise building FMC_AP2AP_AP1.uvproj can get “file not found” error. In FMC_AP2AP_AP1.uvproj, “ap1_image.s” includes image FMC_AP2AP_AP2.bin, that is, FMC_AP2AP_AP2.bin image actually embeds in the data session of FMC_AP2AP_AP1.bin. On run time, FMC_AP2AP_AP1.bin will load and program FMC_AP2AP_AP2.bin to the corresponding flash address.

The execution address of FMC_AP2AP_AP2.bin is 0x4000, and the execution address of FMC_AP2AP_AP1.bin is 0x0. It is determined by R/O base setting of Linker. The following figure shows where to set the linker script in Keil® project.



On system reset, FMC_AP2AP_AP1.bin located at address 0x0 will get execution. It can then jump to FMC_AP2AP_AP2.bin via a sub-routine call, or a branch. Before jumping to FMC_AP2AP_AP2.bin, it issues the “Vector Remap” FMC command to map flash page 0x4000 to memory address 0x0, and then settle the stack pointer. It then obtains the reset handler address of FMC_AP2AP_AP2.bin and cast this address to a function pointer. Now FMC_AP2AP_AP1.bin can jump to FMC_AP2AP_AP2.bin by calling the function pointer. After branch to FMC_AP2AP_AP2.bin, it can branch back to FMC_AP2AP_AP1.bin in the same way but set “Vector Remap” as 0x0 to remap flash page 0x0 back to memory address 0x0. Note that “Vector Remap” can always remap only one flash page. Once a new remapping page issued, the mapping address of old “Vector Remap” flash page turns to naturally mapped address. For example, the naturally mapped address of flash address 0x0 is memory address 0x0, while the naturally mapped address of flash address 0x4000 is memory address 0x4000

1.3 Demo Result

```
+----------------------------------------+
|    Program FMC_AP2AP_AP1 is running    |
+----------------------------------------+
Image FMC_AP2AP_AP2.bin not found!
Program FMC_AP2AP_AP2.bin to APROM.
The starting address is 0x4000.
Program image to flash address 0x4000...OK.
Verify ...OK.
Success.

Branch to run FMC_AP2AP_AP2.bin?
```

```
1-Yes
2-No

+------------------------------+
|   FMC_AP2AP_AP2 is running    |
+------------------------------+

Do you want to branch to FMC_AP2AP_AP1.bin?
1-Yes
2-No

+--------------------------------------+
|    Program FMC_AP2AP_AP1 is running    |
+--------------------------------------+
FMC_AP2AP_AP2.bin found.

Branch to run FMC_AP2AP_AP2.bin?
1-Yes
2-No
```

2 Code Description

ap1_image.s includes FMC_AP2AP_AP2.bin:

```
    AREA _image, DATA, READONLY

    EXPORT  loaderImage1Base              ; export the base address of image
    EXPORT  loaderImage1Limit             ; export the end address of image

    ALIGN   4
            loaderImage1Base
        INCBIN ./obj/FMC_AP2AP_AP2.bin    ; Add image to DATA section
    loaderImage1Limit

        END
```

Program FMC_AP2AP_AP2.bin to flash address 0x4000:

```
    if (FMC_Read(USER_AP2_ENTRY) == 0xFFFFFFFF)
    {
        printf("Image FMC_AP2AP_AP2.bin not found!\r\nProgram FMC_AP2AP_AP2.bin to
                   APROM.\r\nThe starting address is 0x%x.\r\n", USER_AP2_ENTRY);
        FMC_ENABLE_AP_UPDATE();

        /* loaderImage1Base and loaderImage1Limit are defined in ap1_image.s */
        if (load_image_to_flash((uint32_t)&loaderImage1Base, (uint32_t)&loaderImage1Limit,
             USER_AP2_ENTRY, USER_AP2_MAX_SIZE) != 0)
        {
            printf("Failed to program FMC_AP2AP_AP2.bin!\r\n");
            return -1;
        }
        else
        {
             printf("Success.\r\n");
        }
        FMC_DISABLE_AP_UPDATE();
    }
```

```
static int  load_image_to_flash(uint32_t image_base, uint32_t image_limit, uint32_t
                                flash_addr, uint32_t max_size)
{
    uint32_t   i, j, u32Data, u32ImageSize, *pu32Loader;
    u32ImageSize = image_limit - image_base;

    ...

    printf("Program image to flash address 0x%x...", flash_addr);
    pu32Loader = (uint32_t *)image_base;

    for (i = 0; i < u32ImageSize; i += FMC_FLASH_PAGE_SIZE)
    {
        if (FMC_Erase(flash_addr + i))
        {
            printf("Erase failed on 0x%x\r\n", flash_addr + i);
            return -1;
        }
        for (j = 0; j < FMC_FLASH_PAGE_SIZE; j += 4)
        {
            FMC_Write(flash_addr + i + j, pu32Loader[(i + j) / 4]);
        }
    }
    printf("OK.\r\n");
    printf("Verify ...");

    /* Verify loader */
    for (i = 0; i < u32ImageSize; i += FMC_FLASH_PAGE_SIZE)
    {
        for (j = 0; j < FMC_FLASH_PAGE_SIZE; j += 4)
        {
            u32Data = FMC_Read(flash_addr + i + j);
            if (u32Data != pu32Loader[(i + j) / 4])
            {
                printf("data mismatch on 0x%x, [0x%x], [0x%x]\r\n", flash_addr + i + j,
                        u32Data, pu32Loader[(i + j) / 4]);
                return -1;
            }

            if (i + j >= u32ImageSize)
                break;
```

```
        }
    }
    printf("OK.\r\n");
    return 0;
}
```

Jump to another application:

```
__asm __set_SP(uint32_t _sp)
{
    MSR MSP, r0
    BX lr
}

__INLINE void BranchTo(uint32_t u32Address)
{
    FUNC_PTR        *func;
    FMC_SetVectorPageAddr(u32Address);     /* set Vector Remap to u32Address */
    func = (FUNC_PTR *)(*(uint32_t *)(u32Address + 4));  /* get reset handler address */
    __set_SP(*(uint32_t *)u32Address);      /* settle stack pointer register */
    func();
}
```

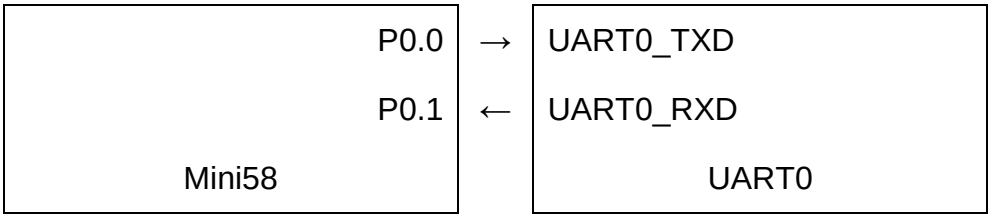
3 Software and Hardware Environment

Software Environment

- BSP version
  - Mini58 Series CMSIS BSP v3.00.003
- IDE version
  - Keil uVersion 5.26

Hardware Environment

- Circuit components
  - NuTiny-EVB-MINI58-LQFP48 V1.0 board
- Diagram



4 Directory Information

- EC_MINI58_FMC_AP2AP_V1.00
  - Library — Sample code header and source files
    - CMSIS — Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
    - Device — CMSIS compliant device header file
    - StdDriver — All peripheral driver header and source files
  - SampleCode
    - ExampleCode — Source file of example code

5 How to Execute Example Code

1. Browsing into sample code folder by 4.Directory Information and double click FMC_AP2AP_AP2.uvproj and build it.
2. Double click FMC_AP2AP_AP1.uvproj.
3. Enter Keil compile mode
   a. Build
   b. Download
   c. Start/Stop debug session
4. Enter debug mode
   a. Run

6 Revision History

Date	Revision	Description
July 02, 2019	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.